WHAT IS CLAIMED IS:

1. A computer-implemented method for improving memory usage for a computer program, comprising:

(a) reading a trace that describes data accesses performed by the computer program;

(b) analyzing the trace to discover data access patterns; and

(c) building a grammar that represents repetitively occurring sequences of the data access patterns.

2. The method of claim 1 further comprising constructing a directed acyclic graph based on the grammar.

3. The method of claim 1, further comprising removing stack references from the trace prior to building the grammar.

4. The method of claim 1, wherein the trace includes heap references, and wherein the method further comprises transforming related heap references in the trace to map to a single identifier.

5. The method of claim 4, wherein the single identifier comprises a starting memory address of a memory block allocated by the program prior to the program making the heap references.

6. The method of claim 1, further comprising:

(d) constructing a sequence of data accesses in the trace; and

(e) identifying the sequence of data accesses as a repetitively occurring sequence if a cost of accessing data in the sequence of data accesses is greater than a selectable threshold.

7. The method of claim 6, wherein computing the cost comprises multiplying a number of times the sequence occurs in the trace by a number of data accesses in the sequence.

8. The method of claim 1, wherein reading the trace further comprises accessing a non-volatile storage device.

9. A computer-implemented method for improving memory usage for a computer program, comprising:

(a) receiving a data access request;

(b) using information about a data access sequence to determine if the data access request refers to part of a frequently occurring data access sequence; and

(c) if the data access request refers to part of the frequently occurring data access sequence, placing data from the data access request in a memory location;

10. The method of claim 9, wherein the memory location comprises a location in a cache memory.

11. The method of claim 10, wherein placing the data from the data access further comprises:

(i) selecting a cache memory address such that a cache conflict with other data accesses of the frequently occurring data access sequence is avoided; and

(ii) placing the data in the selected cache memory address.

12. The method of claim 9, wherein the placing of the data from the data access request in memory further comprises collocating the data from the data access with other data in memory from the frequently occurring data sequence.

13. A computer-readable medium having computer-executable instruction for improving memory usage for a computer program, comprising:

(a) receiving a data access request; and

(b) using information about a data access sequence to determine if the data access request refers to an element of a frequently requested data access sequence.

14. The method of claim 13 further comprising

(a) if the data access request refers to the element of the frequently requested data access sequence, using timing information to determine if a second element from the frequently requested data access sequence can be retrieved in time for the computer program to use without waiting; and

(b) if the second element can be retrieved in time, pre-fetching the second element to memory.

15. The method of claim 14, wherein the pre-fetching occurs after a delay.

16. The method of claim 14, wherein the second element is pre-fetched into main memory.

17. The method of claim 13, further comprising if the data access request is a request for an element of a frequently requested data access sequence:

(a) retrieving the element; and

(b) placing the element in a memory location.

18. The method of claim 17, wherein the memory comprises cache memory.

19. The method of claim 18, wherein placing the element in the cache memory comprises storing the element in cache memory such that a cache conflict with a second element in the frequently requested data access sequence is avoided.

20. The method of claim 17, wherein the memory comprises a RAM.

21. The method of claim 20, wherein placing the element in the RAM comprises collocating the element in the RAM with a second element of the frequently requested data access sequence.

22. A computer-implemented method for analyzing data access patterns for a computer program, comprising:

(a) selecting a sequence of consecutive data accesses from a trace;

(b) determining if a cost of accessing the sequence of consecutive data accesses is greater than a threshold; and

(c) if the cost is greater than the threshold, identifying the sequence of consecutive data accesses as a hot data block.

23. The method of claim 22, further comprising:

(a) if the sequence of consecutive data accesses is identified as a hot data block, collocating in memory data associated with the data accesses during a copying phase of a garbage collector.

24. The method of claim 22, wherein the cost comprises a number of elements in the sequence of consecutive data accesses multiplied by a number of times the sequence of consecutive data accesses repeats in the trace.

25. The method of claim 22, wherein the sequence of data accesses includes a fewest number of data accesses while still satisfying the threshold.

26. A computer-implemented method for improving memory usage for a computer program, comprising:

(a) receiving data accesses as the data accesses are performed by the computer program;

(b) building a grammar that represents a repetitively occurring sequence of the data accesses.

27. The method of claim 26, further comprising constructing a directed acyclic graph based on the grammar.

28. The method of claim 26, further comprising removing stack references from the data accesses such that they are not included in the grammar.

29. The method of claim 26, wherein the data accesses include heap references, and wherein the method further comprises transforming related heap references to map to a single identifier.

30. The method of claim 27, wherein the single identifier comprises a starting memory address of a memory block allocated by the program prior to the program making the heap references.